CS 2302 Data Structures

Spring 2019

MW 10:30-11:50 in CCSB 1.0202

LAB # 5

Claudio Angel Garcia

Instructor: Olac Fuentes

TA: Anindita Nath, Maliheh Zargaran

IA: Eduardo Lara

Peer leader: Erick Macik

Introduction

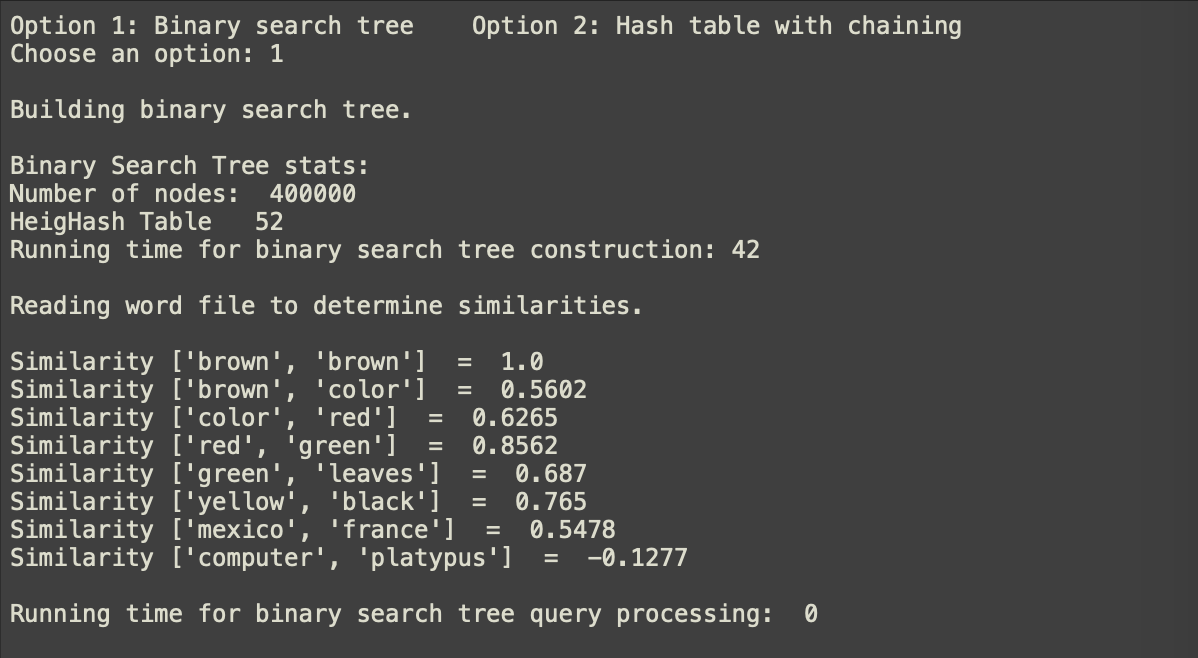
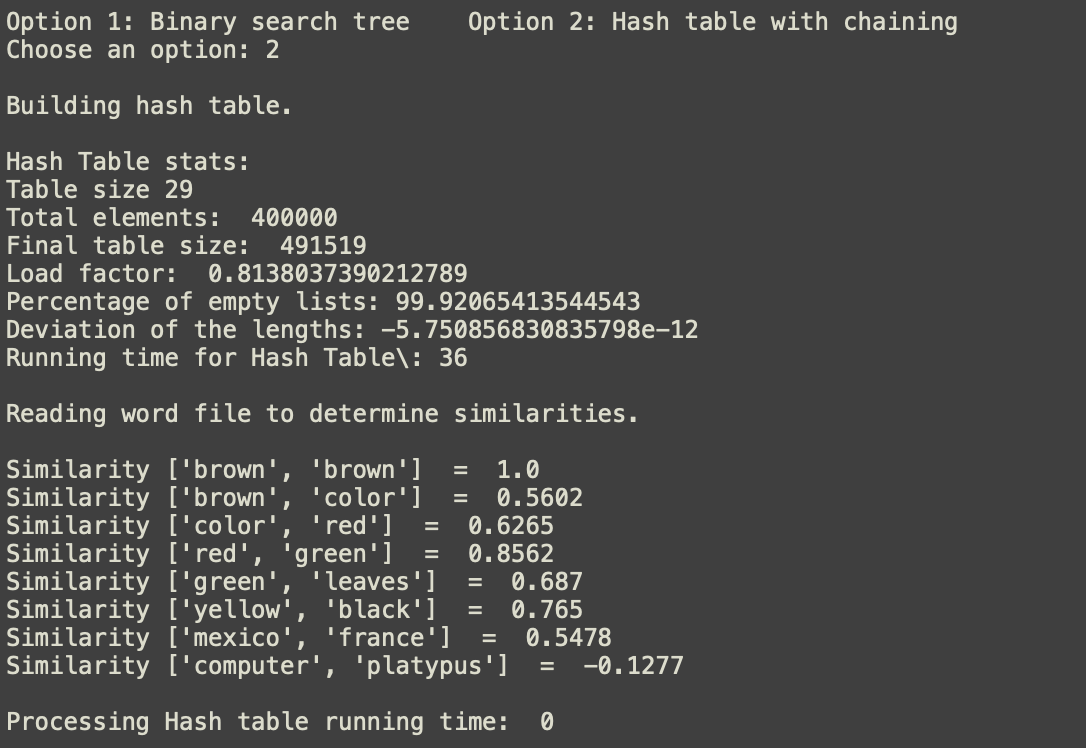
We are tasked to do a program that reads a text document that will contain two words in a line in form of a list. After the program reads the text it will compare the words in the line and return a number depending on their similarities given by an equation between w0 and w1. Before need this, we need to ask the user which implementation of this they want, choosing between a binary search tree or a hash table that uses chaining this to retrieve the embeddings given their corresponding words. When we have the two different implementation for this, we will compare their running times to retrieve word embeddings to enable the comparison of two given words.

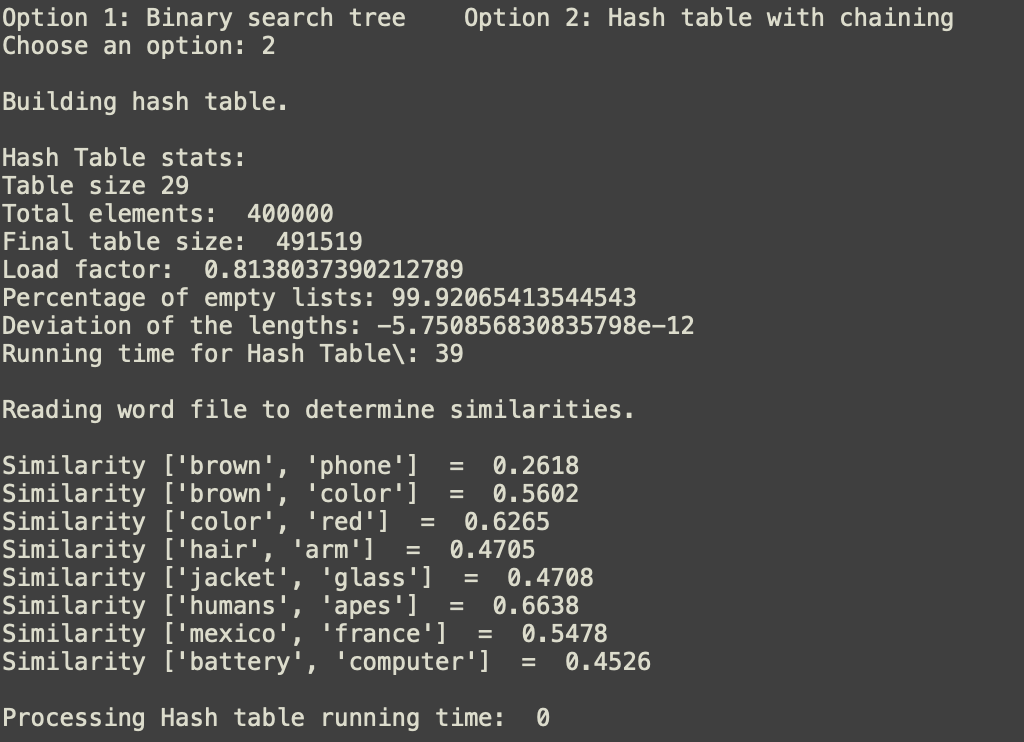
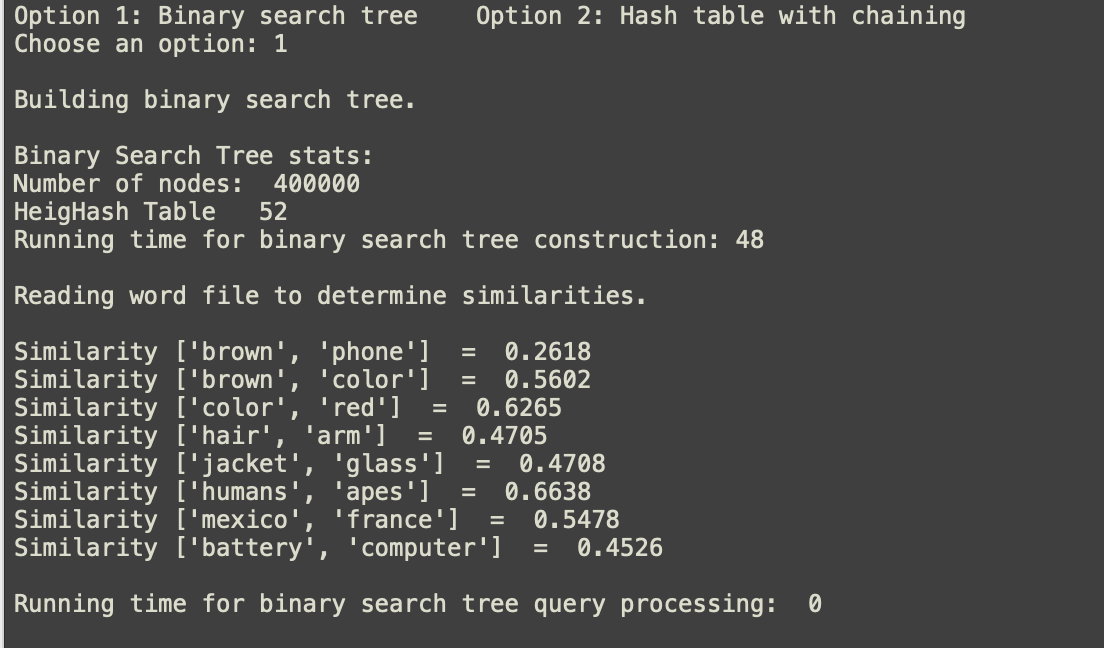
Solution

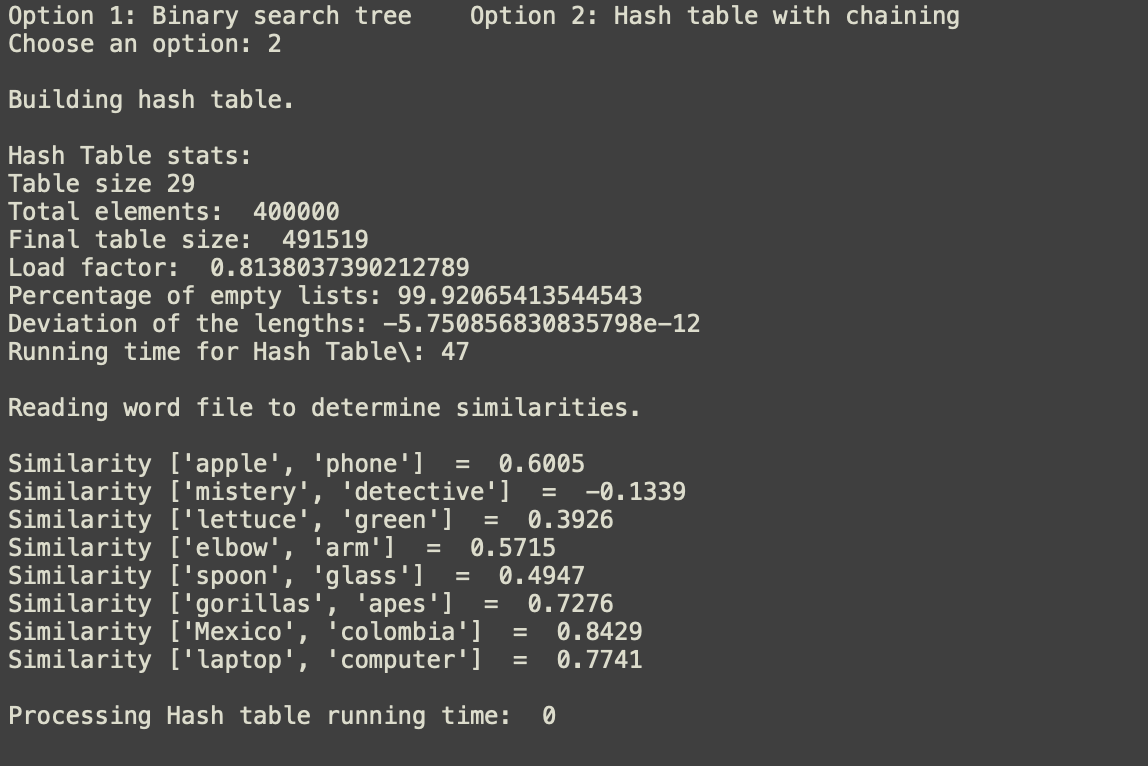
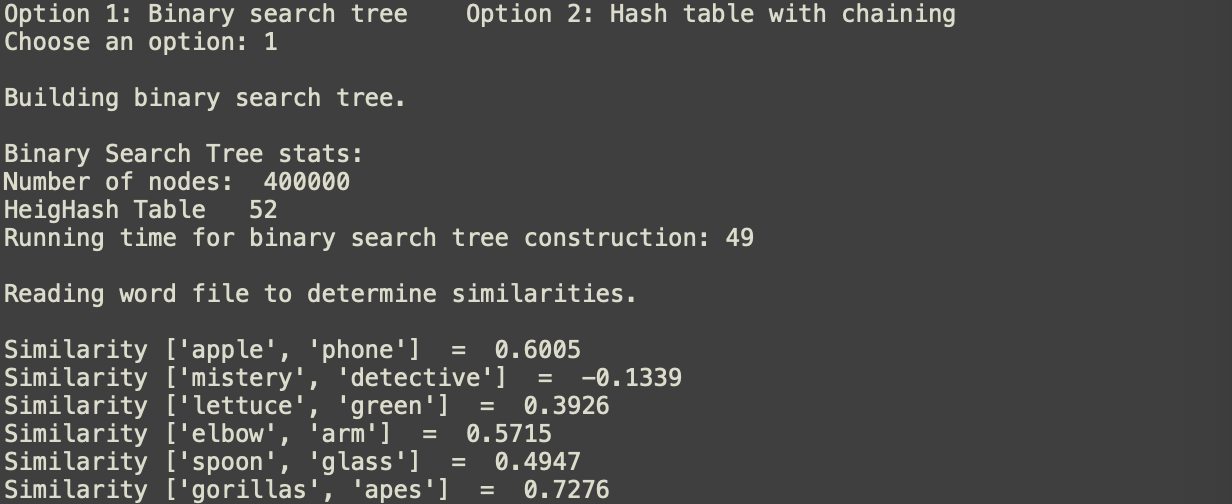
First, we have to ask the user which table implementation they want so we just store the input in a variable that we later use in an if statement. Next is the read file while we construct the hash table and the binary search tree with the text file. For the hash table first we create the table with size 29 then we read line by line the file and we store that I a variable data that we later insert in the hash using chaining for solving the collision, then we print all the information required. Then, we do the same for the word paring, then we store the first and second word in two different variables and with that we can compute the similarity.

For the Binary Search Table we read the file and do a line split and store that in a data variable, then we insert the word arrays into T plus the embeddings. After that, we print all then information required. Next, the word pairs and compute the similarities, first we store the words in two different string variables and we compute the similarities in a for loop.

Experimentation







Conclusion

What I’ve learned in this lab is a way to implement hash tables and binary search trees for a real life usage which would be a part of Natural Language Processing. Thanks to the lab now I have more experience and little but of more knowledge about Artificial Intelligence and also about ways these two very different data structures.

Appendix

import numpy as np

import time

import math

class HashTableC(object):

# Builds a hash table of size 'size'

# Item is a list of (initially empty) lists

# Constructor

def \_\_init\_\_(self, size):

self.item = []

self.num\_items = 0

for i in range(size):

self.item.append([])

class BST(object):

# Constructor

def \_\_init\_\_(self, item=[], left=None, right=None):

self.item = item

self.left = left

self.right = right

############################# Hash Tables methods ##############################

# Build HashTable

def buildHash(f, f2):

print("\nBuilding hash table.\n")

print("Hash Table stats:")

H = HashTableC(29) # create Hash Table of length 17

print("Table size", len(H.item))

start = int(time.time()) # starting time

for line in f: # read line by line, glove

data = line.split(' ')

H = InsertC(H, data) # insert data

end = int(time.time()) # ending time

print("Total elements: ", H.num\_items)

print("Final table size: ", len(H.item))

print("Load factor: ", H.num\_items / len(H.item))

c, d = numEmpty(H)

print("Percentage of empty lists:", c / len(H.item) \* 100)

print("Deviation of the lengths:", d)

print("Running time for Hash Table\:", (end - start))

print("\nReading word file to determine similarities.\n")

start = int(time.time())

for line2 in f2: # read line by line, word\_pair

data2 = line2.split(',')

str1 = FindC(H, data2[0])

str2 = FindC(H, data2[1]) # return array if string found

print("Similarity", data2[0:2], " = ",

round(np.sum(str1 \* str2) / (math.sqrt(np.sum(str1 \* str1)) \* math.sqrt(np.sum(str2 \* str2))),

4)) # compute the similarity

end = int(time.time()) # ending time

print("\nProcessing Hash table running time: ", (end - start))

# Hash Table return the number of empty list and standard deviation of lengths of lists

def numEmpty(H):

c = 0

m = H.num\_items / len(H.item)

k = 0

for a in H.item:

k += len(a) - m

if a == []: # [] found

c += 1

return c, (1 / len(H.item)) \* k

# Hash Table double the size of hashtable

def doubleHashSize(H):

H2 = HashTableC(2 \* len(H.item) + 1) # size = 2\*length+1

for a in H.item: # traverse table

if a != []: # not empty

for i in a: # traverse node since chaining

H2.item[h(i[0], len(H2.item))].append([i[0], i[1]])

H2.num\_items += 1

return H2

# Hash Table insert key, Inserts k in appropriate bucket list

def InsertC(H, k):

# Does nothing if k is already in the table

if H.num\_items // len(H.item) == 1: # recize table

H = doubleHashSize(H)

b = h(k[0], len(H.item)) # get the right index

H.item[b].append([k[0],np.array(k[1:]).astype(np.float)])

H.num\_items += 1 # keep up with elements

return H

# Hash Table return the index to insert

def h(s, n):

r = 0

for c in s:

r = (r \* n + ord(c)) % n

return r

# Hash Table find k and return array if found, Returns bucket (b) and index (i)

def FindC(H, k):

# If k is not in table, i == -1

b = h(k, len(H.item)) # get index

for i in range(len(H.item[b])): # traverse the node

if H.item[b][i][0] == k: # found

return H.item[b][i][1] # return array

return -1

############################# Binary Search Tree methods ##############################

# Build Binary Search tree

def buildBST(f, f2):

print("\nBuilding binary search tree.\n")

T = None

start = int(time.time()) # starting time

for line in f: # get line by line

data = line.split(' ') # array separated by ' '

T = Insert(T, [data[0], np.array(data[1:]).astype(np.float)]) # insert word + embeddings

end = int(time.time()) # ending time

print("Binary Search Tree stats:")

print("Number of nodes: ", nodesInTree(T)) # number of nodes

print("HeigHash Table ", heightTree(T)) # number of height

print("Running time for binary search tree construction:", (end - start))

print("\nReading word file to determine similarities.\n")

start = int(time.time()) # starting time

for line2 in f2: # word pairs

data2 = line2.split(',') # words pair separated by ','

str1 = searchInTree(T, data2[0]) # search the 1st word, return array

str2 = searchInTree(T, data2[1]) # search the 2nd word, return array

print("Similarity", data2[0:2], " = ",round(np.sum(str1 \* str2) / (math.sqrt(np.sum(str1 \* str1)) \* math.sqrt(np.sum(str2 \* str2))),4)) # compute the similarity

end = int(time.time()) # ending time

print("\nRunning time for binary search tree query processing: ", (end - start))

#Bst: insert newitem into T

def Insert(T, newItem):

if T == None:

T = BST(newItem)

elif T.item[0] > newItem[0]:

T.left = Insert(T.left, newItem)

else:

T.right = Insert(T.right, newItem)

return T

#Bst: search a string in the tree T, return node with the same number if it

#was found, None if not found

def searchInTree(T, k):

temp = T # temporary variable for T

while temp is not None: # iterate through necessary nodes

if temp.item[0] == k: # found

temp.item[1]

return temp.item[1]

elif temp.item[0] > k: # smaller

temp = temp.left

else: # larger

temp = temp.right

return None # not found

#Bst: count the number of nodes in T

def nodesInTree(T):

if T is not None:

return 1 + nodesInTree(T.left) + nodesInTree(T.right)

return 0

#Bst: find the height of a tree

def heightTree(T):

if T is not None: # base case

return (1 + max([(heightTree(T.left)), heightTree(T.right)])) # 1 + max number

else:

return -1

ans = '1'

#while ans == '1' or ans == '2':

ans = input("Option 1: Binary search tree Option 2: Hash table with chaining\nChoose an option: ")

f = open('glove.6B.50d.txt', encoding='utf-8')

f2 = open('WordList.txt', encoding='utf-8')

if ans == '1': # binary search tree

buildBST(f,f2)

elif ans == '2': # hash table

buildHash(f, f2)

f.close()

f2.close()

print()

I certify that this project is entirely my own work. I wrote, debugged, and tested the code being presented, performed the experiments, and wrote the report. I also certify that I did not share my code or report or provided inappropriate assistance to any student in the class.

* Claudio Garcia